4.4 VGG16

Data Preprocess of VGG16

In [ ]:

```
import os
import numpy as np
from os import listdir
from imageio import imread
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.utils.image_utils import img_to_array
import keras
import PIL
import matplotlib.pyplot as plt
```

In [ ]:

```
# Settings
num_classes = 10
test_size = 0.2
```

Read Image and Convert to 3D Array

In [ ]:

```
def get_img(data_path):
  ## Getting image array from path:
  img = PIL.Image.open(data_path)
  img = img.convert("L")
  img = img_to_array(img)
  img = np.resize(img, (100, 100, 3))
  return img
```

Get dataset from picture and then split to train and test set

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
dataset_path = "/content/drive/MyDrive/Dataset"

## Getting all data from data path
labels = sorted(listdir(dataset_path))
X = []
Y = []
for i, label in enumerate(labels):
  data_path = dataset_path + "/" + label

  for data in listdir(data_path):
    img = get_img(data_path + "/" + data)
    X.append(img)
    Y.append(i)
## create dataset
X = 1 - np.array(X).astype("float32") /255
Y = np.array(Y).astype("float32")
Y = to_categorical(Y, num_classes)

X, X_test, Y, Y_test = train_test_split(X, Y, test_size=test_size, random_state = 42)
print(X.shape)
print(X_test.shape)
print(Y.shape)
print(Y_test.shape)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
t("/content/drive", force_remount=True).
(1649, 100, 100, 3)
(413, 100, 100, 3)
(1649, 10)
(413, 10)
```

In [ ]:

```
import tensorflow as tf
from numpy.random import seed
seed(1)
tf.random.set_seed(123)
```

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
import sklearn as sk
import time
from keras.datasets import mnist
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Flatten
from keras import optimizers
from keras import backend as K
from keras import regularizers
from keras import initializers
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
import math
from keras import applications
```

In [ ]:

```
img_height = 100
img_width = 100
```

VGG16

Learning rate has been adjusted between le-3, le-4, le-5, le-6, the result is le-5 can provide the best performance. Drop out rate of 0.3, 0.4, 0.5 has been tried and ultimately 0.4 has a relativley good performance. Initially epoch has been set to 10 but the performance was not pretty well. By increasing to 50, the model has been trained into the accuracy of 98% GlobalAveragePooling2D()for creating feature map for each category of the model and unfreezing the base model and retrain the whole model for fine-tuning has been applied in all transfer learning model.

In [ ]:

```
from tensorflow.keras.applications import VGG16
#base_model = VGG16(include_top = False, weights = 'imagenet',input_shape=(100,100,3))
base_model = keras.applications.VGG16(
    weights='imagenet',
    input_shape=(img_height, img_width, 3),
    include_top=False)
#base_model.summary()
```

In [ ]:

```
base_model.trainable = False
```

In [ ]:

```
inputs = keras.Input(shape=(img_height, img_width, 3))
```

In [ ]:

```
x=tf.keras.applications.vgg16.preprocess_input(
    inputs, data_format=None
)
```

In [ ]:

```
x = base_model(x, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.4)(x)
outputs = keras.layers.Dense(10)(x)

model = keras.Model(inputs, outputs)
```

In [ ]:

```
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_12 (InputLayer)       [(None, 100, 100, 3)]     0

 tf.__operators__.getitem_4   (None, 100, 100, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_4 (TFOpLambd  (None, 100, 100, 3)      0
 a)

 vgg16 (Functional)          (None, 3, 3, 512)         14714688

 global_average_pooling2d_4   (None, 512)              0
 (GlobalAveragePooling2D)

 dropout_3 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 14,719,818
Trainable params: 5,130
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [ ]:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
model.fit(X, Y, epochs=3, validation_data=(X_test,Y_test))
```

```
Epoch 1/3
52/52 [==============================] - 4s 58ms/step - loss: 4.3663 - accuracy: 0.1049
- val_loss: 2.3922 - val_accuracy: 0.0969
Epoch 2/3
52/52 [==============================] - 2s 48ms/step - loss: 2.8542 - accuracy: 0.1104
- val_loss: 2.3356 - val_accuracy: 0.0847
Epoch 3/3
52/52 [==============================] - 3s 48ms/step - loss: 2.5525 - accuracy: 0.1013
- val_loss: 2.3267 - val_accuracy: 0.0847
```

Out[ ]:

```
<keras.callbacks.History at 0x7fbe0732ae20>
```

In [ ]:

```
# fine-tuning
base_model.trainable = True
model.summary()

model.compile(
    optimizer=keras.optimizers.Adam(1e-5),  # Low learning rate
    loss=keras.losses.CategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)

epochs = 50
model.fit(X, Y, epochs=epochs, validation_data=(X_test,Y_test))
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_12 (InputLayer)       [(None, 100, 100, 3)]     0         
                                                                 
 tf.__operators__.getitem_4   (None, 100, 100, 3)      0         
 (SlicingOpLambda)                                               
                                                                 
 tf.nn.bias_add_4 (TFOpLambd  (None, 100, 100, 3)      0         
 a)                                                              
                                                                 
 vgg16 (Functional)          (None, 3, 3, 512)         14714688  
                                                                 
 global_average_pooling2d_4   (None, 512)              0         
 (GlobalAveragePooling2D)                                        
                                                                 
 dropout_3 (Dropout)         (None, 512)               0         
                                                                 
 dense_3 (Dense)             (None, 10)                5130      
                                                                 
=================================================================
Total params: 14,719,818
Trainable params: 14,719,818
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
52/52 [==============================] - 7s 121ms/step - loss: 2.3724 - accuracy: 0.0958
- val_loss: 2.3056 - val_accuracy: 0.1211
Epoch 2/50
52/52 [==============================] - 6s 117ms/step - loss: 2.3099 - accuracy: 0.1079
- val_loss: 2.2978 - val_accuracy: 0.1162
Epoch 3/50
52/52 [==============================] - 6s 118ms/step - loss: 2.3116 - accuracy: 0.1031
- val_loss: 2.2990 - val_accuracy: 0.1356
Epoch 4/50
52/52 [==============================] - 6s 119ms/step - loss: 2.3058 - accuracy: 0.1164
- val_loss: 2.2979 - val_accuracy: 0.0920
Epoch 5/50
52/52 [==============================] - 6s 121ms/step - loss: 2.2945 - accuracy: 0.1164
- val_loss: 2.3042 - val_accuracy: 0.0775
Epoch 6/50
52/52 [==============================] - 6s 122ms/step - loss: 2.2873 - accuracy: 0.1128
- val_loss: 2.2932 - val_accuracy: 0.1525
Epoch 7/50
52/52 [==============================] - 6s 121ms/step - loss: 2.2633 - accuracy: 0.1407
- val_loss: 2.2473 - val_accuracy: 0.1550
Epoch 8/50
52/52 [==============================] - 6s 119ms/step - loss: 2.1726 - accuracy: 0.1686
- val_loss: 1.9856 - val_accuracy: 0.2373
Epoch 9/50
52/52 [==============================] - 6s 119ms/step - loss: 1.9864 - accuracy: 0.2608
- val_loss: 1.8227 - val_accuracy: 0.2421
Epoch 10/50
52/52 [==============================] - 6s 118ms/step - loss: 1.4740 - accuracy: 0.4463
- val_loss: 0.9207 - val_accuracy: 0.6998
Epoch 11/50
```

```
52/52 [==============================] - 6s 118ms/step - loss: 1.1139 - accuracy: 0.5919
- val_loss: 0.8084 - val_accuracy: 0.7191
Epoch 12/50
52/52 [==============================] - 6s 118ms/step - loss: 0.8675 - accuracy: 0.6871
- val_loss: 0.6650 - val_accuracy: 0.7554
Epoch 13/50
52/52 [==============================] - 6s 118ms/step - loss: 0.7169 - accuracy: 0.7374
- val_loss: 0.5854 - val_accuracy: 0.8160
Epoch 14/50
52/52 [==============================] - 6s 118ms/step - loss: 0.6123 - accuracy: 0.7829
- val_loss: 0.3710 - val_accuracy: 0.8765
Epoch 15/50
52/52 [==============================] - 6s 119ms/step - loss: 0.4866 - accuracy: 0.8369
- val_loss: 0.4846 - val_accuracy: 0.8450
Epoch 16/50
52/52 [==============================] - 6s 121ms/step - loss: 0.5350 - accuracy: 0.8114
- val_loss: 0.5098 - val_accuracy: 0.8232
Epoch 17/50
52/52 [==============================] - 6s 120ms/step - loss: 0.4257 - accuracy: 0.8466
- val_loss: 0.4723 - val_accuracy: 0.8499
Epoch 18/50
52/52 [==============================] - 6s 120ms/step - loss: 0.3911 - accuracy: 0.8629
- val_loss: 0.3960 - val_accuracy: 0.8741
Epoch 19/50
52/52 [==============================] - 6s 121ms/step - loss: 0.3198 - accuracy: 0.8896
- val_loss: 0.2608 - val_accuracy: 0.9249
Epoch 20/50
52/52 [==============================] - 6s 121ms/step - loss: 0.3174 - accuracy: 0.8896
- val_loss: 0.3662 - val_accuracy: 0.8789
Epoch 21/50
52/52 [==============================] - 6s 121ms/step - loss: 0.3209 - accuracy: 0.8890
- val_loss: 0.2470 - val_accuracy: 0.9322
Epoch 22/50
52/52 [==============================] - 6s 120ms/step - loss: 0.2785 - accuracy: 0.9078
- val_loss: 0.2448 - val_accuracy: 0.9298
Epoch 23/50
52/52 [==============================] - 6s 120ms/step - loss: 0.1746 - accuracy: 0.9351
- val_loss: 0.2101 - val_accuracy: 0.9443
Epoch 24/50
52/52 [==============================] - 6s 120ms/step - loss: 0.2193 - accuracy: 0.9200
- val_loss: 0.2816 - val_accuracy: 0.9153
Epoch 25/50
52/52 [==============================] - 6s 120ms/step - loss: 0.2029 - accuracy: 0.9303
- val_loss: 0.1815 - val_accuracy: 0.9540
Epoch 26/50
52/52 [==============================] - 6s 120ms/step - loss: 0.1532 - accuracy: 0.9472
- val_loss: 0.2500 - val_accuracy: 0.9274
Epoch 27/50
52/52 [==============================] - 6s 119ms/step - loss: 0.1755 - accuracy: 0.9466
- val_loss: 0.2564 - val_accuracy: 0.9346
Epoch 28/50
52/52 [==============================] - 6s 119ms/step - loss: 0.1698 - accuracy: 0.9388
- val_loss: 0.1778 - val_accuracy: 0.9467
Epoch 29/50
52/52 [==============================] - 6s 119ms/step - loss: 0.1109 - accuracy: 0.9636
- val_loss: 0.2167 - val_accuracy: 0.9492
Epoch 30/50
52/52 [==============================] - 6s 120ms/step - loss: 0.1336 - accuracy: 0.9521
- val_loss: 0.2513 - val_accuracy: 0.9225
Epoch 31/50
52/52 [==============================] - 6s 119ms/step - loss: 0.1439 - accuracy: 0.9515
- val_loss: 0.1878 - val_accuracy: 0.9564
Epoch 32/50
52/52 [==============================] - 6s 120ms/step - loss: 0.1092 - accuracy: 0.9630
- val_loss: 0.2315 - val_accuracy: 0.9370
```

```
Epoch 33/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0907 - accuracy: 0.9685
- val_loss: 0.2425 - val_accuracy: 0.9370
Epoch 34/50
52/52 [==============================] - 6s 122ms/step - loss: 0.1213 - accuracy: 0.9576
- val_loss: 0.2830 - val_accuracy: 0.9201
Epoch 35/50
52/52 [==============================] - 6s 121ms/step - loss: 0.1059 - accuracy: 0.9630
- val_loss: 0.2733 - val_accuracy: 0.9298
Epoch 36/50
52/52 [==============================] - 6s 120ms/step - loss: 0.1107 - accuracy: 0.9594
- val_loss: 0.1939 - val_accuracy: 0.9492
Epoch 37/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0740 - accuracy: 0.9697
- val_loss: 0.1391 - val_accuracy: 0.9661
Epoch 38/50
52/52 [==============================] - 6s 119ms/step - loss: 0.0795 - accuracy: 0.9721
- val_loss: 0.1908 - val_accuracy: 0.9467
Epoch 39/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0686 - accuracy: 0.9745
- val_loss: 0.2074 - val_accuracy: 0.9492
Epoch 40/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0757 - accuracy: 0.9715
- val_loss: 0.2020 - val_accuracy: 0.9467
Epoch 41/50
52/52 [==============================] - 6s 123ms/step - loss: 0.0877 - accuracy: 0.9691
- val_loss: 0.1446 - val_accuracy: 0.9516
Epoch 42/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0482 - accuracy: 0.9830
- val_loss: 0.1810 - val_accuracy: 0.9564
Epoch 43/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0632 - accuracy: 0.9770
- val_loss: 0.1555 - val_accuracy: 0.9564
Epoch 44/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0484 - accuracy: 0.9848
- val_loss: 0.1387 - val_accuracy: 0.9661
Epoch 45/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0437 - accuracy: 0.9854
- val_loss: 0.3471 - val_accuracy: 0.8959
Epoch 46/50
52/52 [==============================] - 6s 122ms/step - loss: 0.0600 - accuracy: 0.9794
- val_loss: 0.1436 - val_accuracy: 0.9661
Epoch 47/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0369 - accuracy: 0.9861
- val_loss: 0.2532 - val_accuracy: 0.9298
Epoch 48/50
52/52 [==============================] - 6s 119ms/step - loss: 0.0545 - accuracy: 0.9806
- val_loss: 0.1625 - val_accuracy: 0.9637
Epoch 49/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0461 - accuracy: 0.9861
- val_loss: 0.1833 - val_accuracy: 0.9613
Epoch 50/50
52/52 [==============================] - 6s 120ms/step - loss: 0.0377 - accuracy: 0.9873
- val_loss: 0.1540 - val_accuracy: 0.9564
```

```
Out[ ]: <keras.callbacks.History at 0x7fbe075a49d0>
```

Accuracy: 98.73%

By comparing model of transfered learning model with Based on the results from model we built from scratch, the accuracy of fully connected model is 71%, accuracy of CNN model is 80.34%. All the five models of transfer learning have a better performance than those two since transfer learning will include the saving of resources and improve efficiety when training new models with complex layers.